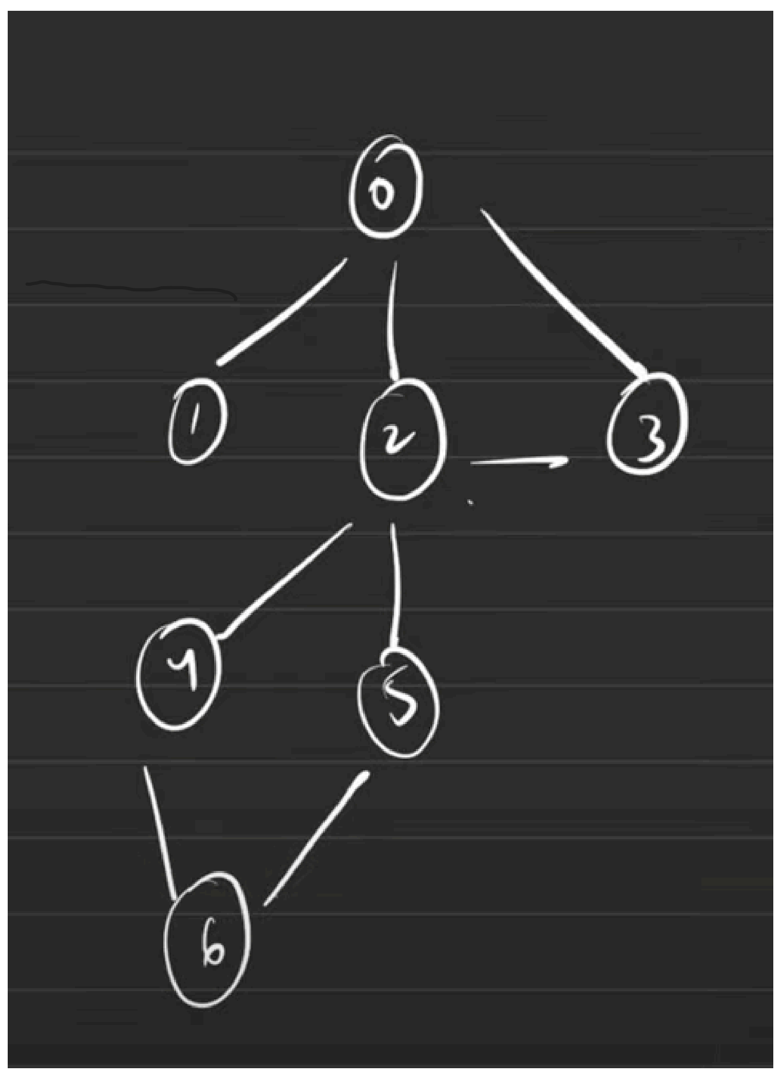


Nodes on whose removal the graph breaks down into components.

E.g:- "0" is one such node, "2" as well

One major difference from bridges is - we were removing edges there and we are removing entire nodes here.

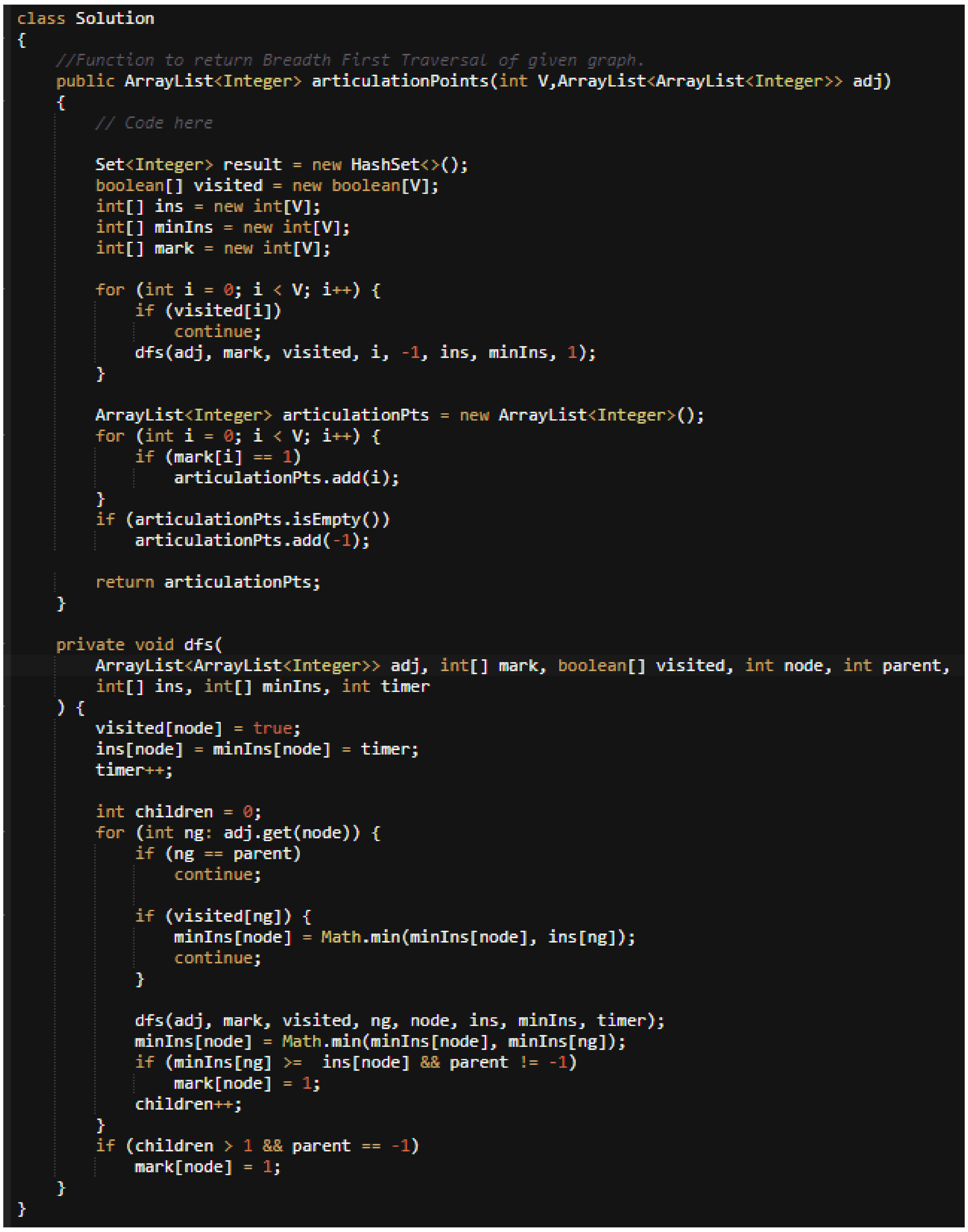
Similar to bridges problem we have two arrays one to store the time of insertion of nodes through DFS and other to store min insertion time.

The minimum insertion time here is only taken from adjacent nodes which are not visited and not parent.

For checking if node is articulation point we can just check if the min insertion time of neighbor is greater than or equal to insertion time of node

For root node we will just check if it has multiple children or not.

Same node can be found as an articulation point multiple times so we can just maintain an hash set to keep track instead of list to prevent duplicates.



```java
class Solution
{
    //Function to return Breadth First Traversal of given graph.
    public ArrayList<Integer> articulationPoints(int V,ArrayList<ArrayList<Integer>> adj)
    {
        // Code here

        Set<Integer> result = new HashSet<>();
        boolean[] visited = new boolean[V];
        int[] ins = new int[V];
        int[] minIns = new int[V];
        int[] mark = new int[V];

        for (int i = 0; i < V; i++) {
            if (visited[i])
                continue;
            dfs(adj, mark, visited, i, -1, ins, minIns, 1);
        }

        ArrayList<Integer> articulationPts = new ArrayList<Integer>();
        for (int i = 0; i < V; i++) {
            if (mark[i] == 1)
                articulationPts.add(i);
        }
        if (articulationPts.isEmpty())
            articulationPts.add(-1);

        return articulationPts;
    }

    private void dfs(
        ArrayList<ArrayList<Integer>> adj, int[] mark, boolean[] visited, int node, int parent,
        int[] ins, int[] minIns, int timer
    ) {
        visited[node] = true;
        ins[node] = minIns[node] = timer;
        timer++;

        int children = 0;
        for (int ng: adj.get(node)) {
            if (ng == parent)
                continue;

            if (visited[ng]) {
                minIns[node] = Math.min(minIns[node], ins[ng]);
                continue;
            }

            dfs(adj, mark, visited, ng, node, ins, minIns, timer);
            minIns[node] = Math.min(minIns[node], minIns[ng]);
            if (minIns[ng] >=  ins[node] && parent != -1)
                mark[node] = 1;
            children++;
        }
        if (children > 1 && parent == -1)
            mark[node] = 1;
    }
}
```

Time Complexity: O(V+2E), where V = no. of vertices, E = no. of edges. It is because the algorithm is just a simple DFS traversal.

Space Complexity: O(3V), where V = no. of vertices. O(3V) is for the three arrays i.e. tin, low, and vis, each of size V.